УДК 510.5+004.046

І.С. Мінтій
Криворізький державний педагогічний університет

МАТЕМАТИЧНІ ОСНОВИ ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

У статті встановлено відповідність між формальною системою λ-числення та функціональним програмуванням; проаналізовано, що застосування функціонального математичного програмування дозволяє описати обчислювальний процес у вигляді абстракцій функцій та їх аплікацій; доведено, що простота та легкість програм дозволяє розглядати можливість використання функціонального математичного програмування у процесі навчання основ програмування.

Ключові слова: *основи програмування, абстрактний матеріал, формальна система, студент, константа, елемент, символ.*

Постановка проблеми. Математична логіка – досить складний предмет. Особливо це стосується таких її розділів, як «Формальні теорії» («формальні системи»). Це пояснюється високим рівнем абстрактності матеріалу. Ці розділи часто вивчаються поверхово: через перерозподіл годин в межах курсу на користь традиційних розділів («Числення висловлень», «Логіка предикатів першого порядку», тощо). Особливо гостро ця проблема постає в процесі навчання студентів інформатичних та фізичних спеціальностей, в яких розділ «Формальні теорії» через брак аудиторного часу може бути навіть повністю винесений на самостійне опрацювання.

Ситуацію можна покращити, використовуючи міжпредметні зв’язки математичної логіки. Так, у встановлено відповідність між формальними численнями та фундаментальними розділами курсу інформатики (таблиця 1) [2, 168].

Таблиця 1

Відповідність між формальними численнями та фундаментальними розділами курсу інформатики

Формальне числення	*Розділ курсу інформатики*
λ-числення Числення комбінаторів	Функціональне програмування
Числення регулярних виразів та регулярних рівностей Прямолінійні граматики, КВ-граматики, КЗ-граматики	Формальний синтаксис мов програмування
Системи і напівсистеми Туе Канонічні і нормальні системи Поста	Продукційне програмування
Кінцеві автомати Рабіна-Скотта МП-автомати Машини Тьюрінга	Формальний синтаксис мов програмування Операційна семантика мов програмування Архітектура обчислювальних систем

Формальне числення	*Розділ курсу інформатики*
Лінійно-обмежені автомати Кінцеві перетворення Рабіна-Скотта Автомати Мілі і Мура	
Логічне числення	Логічне програмування
Числення програм Хоара і Дейкстри	Дедуктивна семантика мов програмування

Мета статті – встановлення відповідності між формальною системою λ-числення та функціональним програмуванням.

Аналіз актуальних досліджень. Питання дослідження фундаментальних основ інформатики (а разом з тим, і проблем вивчення формальної математики) розглядається в працях В. В. Лаптєва, Н. І. Рижової, М. В. Швецького.

Вивченню функціональних мов програмування (ФМП) присвячено праці Х. Абельсона, І. Г. Косової, С. М. Малярчука, Дж. МакКарті, Ю. І. Машбиця, Д. Д. Сассмана, Дж. Сассман, Р. У. Себеста, Й. Сеппянена, І. О. Теплицького, А. Філда, Б. Харві, П. Харрісона, Е. Хювьонена та ін.

Виклад основного матеріалу. Для початку розглянемо означення формальної системи.

Формальна система (числення) – це математична модель, яка задає множину дискретних об’єктів шляхом опису початкових об’єктів і правил утворення нових об’єктів з початкових та вже утворених. Об’єкти формальної системи складаються з неподільних елементів різних видів [2, 93].

Для побудови формальної мови необхідно скористатись якоюсь уже відомою мовою та в термінах цієї мови створити алфавіт і сформулювати правила формальної мови [2, 48]. Алфавіт утворюється шляхом задання символів, що називаються початковими символами мови і є неподільними за таких умов:

а) при утворенні мови ніколи не використовуються їх частини;

б) будь-яку скінченну послідовність початкових символів можна утворити єдиним способом з елементів множини початкових символів.

Скінченна послідовність початкових символів називається формулою. Формули утворюються відповідно до визначених правил. Деякі з формул оголошуються аксіомами. І, нарешті, встановлюються правила виведення (або правила дії, або правила перетворення), згідно з якими з формул безпосередньо виводиться як висновок деяка формула. Скінченна послідовність, яка складається з однієї чи більшого

числа формул називається доведенням, якщо кожна формула в послідовності є або аксіомою або виводиться безпосередньо згідно з одним із правил виведення з попередньо утворених формул послідовності.

Визначаючи формальну систему, необхідно додати ще вимоги ефективності [2, 50]:

1) задання початкових символів повинно бути ефективним в тому сенсі, що повинен існувати метод, який завжди дозволяє ефективно визначити, чи є деякий символ одним з початкових символів послідовності;

2) визначення формули повинно бути ефективним в тому сенсі, що повинен існувати метод, який завжди дозволяє визначити, чи є деяка формула правильно утворена;

3) задання аксіом повинно бути ефективним в тому сенсі, що повинен існувати метод, який завжди дозволяє ефективно визначити, чи є формула аксіомою;

4) правила виведення повинні бути ефективними в тому строгому сенсі, що повинен існувати метод, який завжди дозволяє ефективно визначити, чи виводиться безпосередньо згідно з правилами виведення.

Графічне подання складових формальної системи [2, 168] показано на рис. 1:

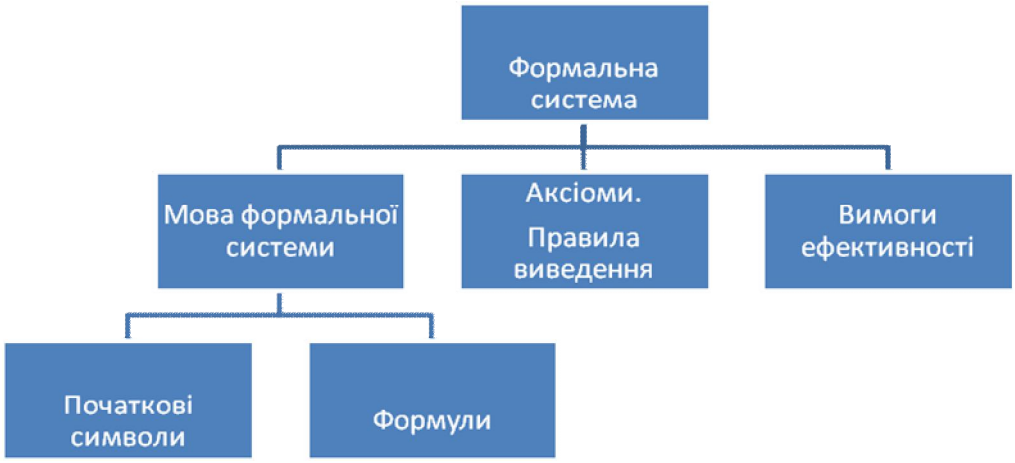

Рис. 1. Складові формальної системи (В. В. Лаптєв, Н. І. Рижова, М. В. Швецький)

На нашу думку, таке подання не відображає взаємозв'язок між елементами формальної системи і вимагає переробки:

Рис. 2. Взаємозв'язок складових формальної системи

Опишемо формальну систему λ-числення згідно з названими правилами та паралельно з цим будемо встановлювати відповідність між λ-численням і ФМП Scheme. Цю мову ми обрали виключно завдяки її простим синтаксису та семантиці.

Алфавіт λ-числення містить такі елементи:

– константи та змінні (позначаються малими буквами латинського алфавіту, можливе використання індексів, наприклад, c1, c2, x, y…);

– вирази або формули (або, інакше, – терми) (позначаються великими літерами латинського алфавіту, можливе використання індексів, наприклад, M, N, ...);

– спеціальні символи («(», «)», «.»).

Алфавіт ФМП Scheme:

– константи та змінні (можуть складатися з великих та малих букв, цифр та спеціальних символів (за виключенням дужок), не повинні починатися з цифри та містити пробіли; наприклад, c1, c2, x, y, Д…);

– елементарні функції для виконання основних арифметичних операцій: +, –, *, / (функції «+» і «*» можуть містити декілька параметрів (у тому числі і більше 2); якщо функція «*» задана без параметрів, вона повертає 1, функція «+» без параметрів повертає 0; при використанні функції «–» всі параметри, окрім першого, віднімаються від першого; ділення схоже на віднімання);

– спеціальні символи: «(», «)», «'», «.» та елементарні предикати «=», «>», «<», «>=», «<=».

У ФМП Scheme використовується префіксна нотація (прямий польський запис). Тобто, інфіксний запис 3 + 4 в префіксній нотації матиме

вигляд (+ 3 4). Scheme, як і більшість ФМП, працює зі списками, що позначаються круглими дужками. Крайній зліва елемент списку – функція, інші елементи – аргументи функції. Тобто, звичний для нас запис f(x, y) має вигляд (f x y). Переваги префіксної нотації:

– префіксна форма запису виразу може містити більше, ніж 2 аргументи:

(+ 2 3 4); (* 5 6 9);

– вона природно розширюється, в результаті чого отримуються вкладені списки (композиції функцій), елементами яких також є списки:

(+ (– 6 3 4)

(* 4 5 2)).

На додачу до елементарних предикатів існують операції логічної композиції, які дозволяють утворювати складені предикати:

– (and (предикат1) … (предикатn));

– (or (предикат1) … (предикатn));

– (not (предикат)).

Так, наприклад, умова приналежності числа x відрізку [a, b] (x Є [a, b]) виглядатиме так:

(and (>= x a)

(<= x b)).

Розглянемо *правила* утворення формул (або, інакше, – λ-виразів):

λ-терми утворюються таким чином:

– будь-яка змінна або константа є λ-термом;

– якщо M, N – довільні λ-терми і x – довільна змінна, то вірно, що:

1) вираз (λx.M) є λ-термом;

2) вираз (MN) є λ-термом.

λ-вираз (λx.M) – означає операцію абстракції (визначення функції M з аргументом x), а λ-вираз (MN) означає операцію аплікації (застосування функції M до аргументу N).

У ФМП Scheme загальна форма запису абстракції для створення функцій:

(lambda (формальні параметри) тіло).

Наприклад, функція інкременту (збільшення аргументу на 1) має вигляд:

(lambda (x) (+ x 1)).

Якщо ми хочемо назвати функцію, яку описує λ-вираз, необхідно використати спеціальну форму define, що пов’язує між собою ім’я функції

та її λ-вираз. Наприклад:

(define inc
(lambda (x)
(+ x 1))).

У цьому випадку λ-вираз скорочується: слово lambda та дужки початку й кінця списку видаляють. У такій формі функція define отримує в якості параметрів дві частини. Перша – прототип виклику функції, який складається з імені функції та списку її формальних параметрів, а друга – тіло функції.

Застосування функції (ілюстрація операції аплікації):

(inc 5). Результат: 6.

Або ж (якщо ми використовуємо безіменну функцію):

((lambda (x) (+ x 1)) 5). Результат: 6.

Для більшої наочності покажемо відповідність між правилами утворення формул в λ-численні та правилами створення функцій в ФМП Scheme (таблиця 2):

Таблиця 2

Відповідність між правилами утворення формул в λ-численні
та правилами створення функцій в ФМП Scheme

	λ-числення	ФМП Scheme
Операція абстракції	(MN)	(M N)
операції аплікації	(λx.M)	(define M (lambda (x) …)

Аксіоми λ-числення:

1. λx.a = λy.[y/x]a;

2. (λx.a)b = [b/x]a.

Символ «=» в λ-численні позначає відношення конвертованості. Конвертованість двох λ-виразів означає, що один λ-вираз може бути перетворений в інший.

Аксіома 1 означає можливість підстановки терма у замість всіх входжень змінної x в λ-вираз a. Стосовно мови програмування дана аксіома трактується так: результат застосування функції не залежить від імен формальних параметрів. Наприклад, результат функцій double1 і double2:

(define (double1 x) (define (double2 y)
(* 2 x)); (* 2 y)).

буде однаковим, якщо їх застосувати до одного й того ж числа:

(double1 4) → результат 4; (double2 4) →результат 4

Аксіома 2 означає можливість редукції (спрощення вигляду) λ-виразу в лівій частині шляхом підстановки b замість всіх входжень змінної x в λ-вираз a.

Розглянемо, наприклад, таку задачу: створити програму, яка повертає суму квадратів двох чисел. Розв'язок можна створивши безпосередньо функцію sum_of_squares:

```
(define (sum_of_squares1 x y)
(+ (* x x)
 (* y y)))
```

А можна створити функцію, яка повертає квадрат аргументу, і, використовуючи її, створити функцію sum_of_squares:

```
(define (square x)
(* x x))
(define (sum_of_squares2 x y)
(+ (square x)
(square y)))
```

У другому випадку ми спростили вигляд функції sum_of_squares шляхом підстановки функції square.

Перевірка:

(sum_of_squares1 2 3) 13;

(sum_of_squares2 2 3) 13.

Інший приклад: створити функцію, яка повертає більше з трьох чисел.

Перший варіант розв'язку (послідовне порівняння одного числа з двома іншими):

```
(define (max_three x y z)
        (cond ((and (>= x y) (>= x z)) x)
              ((and (>= y x) (>= y z)) y)
              ((and (>= z x) (>= z y)) z))).
```

У програмі використана функція розгалуження cond, загальний вигляд якої:

```
        (cond (предикат 1 функція 1)
              (предикат 2 функція 2)
        .....
              (предикатn функціяn)
              (else функція)).
```

Вона складається з символа cond, після якого записані двоелементні

списки, що складаються з предикату та функції. Порядок обчислення такий: спочатку обчислюється предикат1, якщо його значення хибне – обчислюється предикат 2. Так продовжується доти, доки не знайдеться предикат, який має істинне значення. В цьому випадку результатом функції розгалуження cond буде значення відповідної функції. Якщо жоден з предикатів не матиме істинного значення, значення cond не визначено; else – спеціальний символ в заключній вітці cond, що відповідає тотожньо істинній функції. Якщо жоден з предикатів не матиме істинного значення, cond буде мати значення функції, яка знаходиться у вітці else.

У другому варіанті розв'язку використано функцію max, яка повертає більше з двох чисел:

```
(define (max x y)
        (cond (> x y) x)
                (else y))).
```

Це дозволяє можна отримати елегантну, але від цього не менш ефективну форму запису функції max_three:

```
(define (max_three x y z)
(max (max x y) z)).
```

Правила виведення термів, які задають характеристики відношення конвертованості:

1) якщо a=b, то ca=cb.

Мовою програмування, це правило трактується так: результат функції, застосованої до однакових аргументів, буде однаковий. Наприклад:

```
(define a 6).
(define b 6).
```

В результаті таких дій було створено дві функції-константи a і b з однаковим значенням 6.

Застосуємо функцію інкременту до аргументів a і b:

(inc a) → результат 7; (inc b) → результат 7.

2) якщо a=b, то ac=bc.

Тобто, результати однакових функцій, застосованих до одного й того ж аргументу, будуть однакові:

(define (a x) (+ x 1)); (define (b x) (+ x 1)).

В результаті: створено дві однакові функції функції a і b. Тоді:

(a 3) → результат 4; (b 3) → результат 4.

3) якщо a=b, то λx.a= λx.b:

(define a (lambda (x) (+ x 1))) (define b (lambda (x) (+ x 1)))

Однакові функції мають однакові тіла.

4) a=a (рефлексивність);

5) якщо a=b, то b=a (симетричність);

6) якщо a=b і b=c, то a=c (транзитивність).

Висновки.

1. Функціональний підхід до побудови мов програмування дозволяє реалізувати в них основні положення λ-числення.

2. Застосування ФМП дозволяє описати обчислювальний процес у вигляді абстракцій функцій та їх аплікацій.

3. Простота та легкість програм дозволяє розглядати можливість використання ФМП у процесі навчання основам програмування студентів природничо-математичних спеціальностей.

ЛІТЕРАТУРА

1. Информатика: энциклопедический словарь для начинающих / Сост. Д. А. Поспелов. – М. : Педагогика-Пресс, 1994. − 352 с.

2. Лаптев В. В. Методическая теория обучения информатике. Аспекты фундаментальной подготовки / Лаптев В. В., Рыжова Н. И., Швецкий М. В. − СПб. : Изд-во С.-Петерб. ун-та, 2003. − 352 с.

3. Чёрч А. Введение в математическую логику. Т. 1. / Чёрч А. –М. : Изд-во иностранной литературы, 1960. – 486 с.

РЕЗЮМЕ

В статье установлено соответствие между формальной системой λ-исчисления и функциональным программированием; проанализировано, что применение функционального математического программирования позволяет описать вычислительный процесс в виде абстракций функций и их приложений; доказано, что простота и легкость программ позволяет рассматривать возможность использования функционального математического программирования в процессе обучения основам программирования.

Ключевые слова: *основы программирования, абстрактный материал, формальная система, студент, константа, элемент, символ.*

SUMMARY

The article established the correspondence between the formal system of λ-calculus and functional programming, analysis, the use of functional mathematical programming to describe the calculated process as an abstraction functions and their applications, it is proved that simplicity and ease of application to consider the use of mathematical operate programming in teaching students the basics of programming

Key words: *bases of programming, abstract material, formal system, student, a constant element, the symbol.*